2D Drawing


Also see the Java 8 tutorial – [http://www.coreservlets.com/java-8-tutorial/](http://www.coreservlets.com/java-8-tutorial/)

and customized Java training courses (onsite or at public venues) – [http://courses.coreservlets.com/java-training.html](http://courses.coreservlets.com/java-training.html)

Customized Java EE Training: [http://courses.coreservlets.com/](http://courses.coreservlets.com/)

Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Java-related training, email hall@coreservlets.com

Marty is also available for consulting and development support

Taught by lead author of *Core Servlets & JSP*, co-author of *Core JSF* (4th Ed), & this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2.2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  - Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details
Topics in This Section

- Overview
- Drawing Shapes
- Paint Styles
- Transparency
- Using Local Fonts
- Stroke Styles
- Coordinate Transformations
- Requesting Drawing Accuracy

Java 2D

Drawing Shapes

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.
AWT vs. Java 2D Drawing: Overview

**AWT**

```java
public void paint(Graphics g) {
    // Set pen parameters
    g.setColor(someColor);
    g.setFont(someLimitedFont);
    // Draw a shape
    g.drawString(...);
    g.drawLine(...)
    g.drawRect(...);     // outline
    g.fillRect(...);     // solid
    g.drawPolygon(...);  // outline
    g.fillPolygon(...);  // solid
    g.drawOval(...);     // outline
    g.fillOval(...);     // solid
    ...
}
```

**Java 2D**

```java
public void paintComponent(Graphics g) {
    // Clear off-screen bitmap
    super.paintComponent(g);
    // Cast Graphics to Graphics2D
    Graphics2D g2d = (Graphics2D)g;
    // Set pen parameters
    g2d.setPaint(fillColorOrPattern);
    g2d.setStroke(penThicknessOrPattern);
    g2d.setComposite(someAlphaComposite);
    g2d.setFont(anyFont);
    g2d.translate(...);
    g2d.rotate(...);
    g2d.scale(...);
    g2d.shear(...);
    g2d.setTransform(someAffineTransform);
    // Create a Shape object
    SomeShape s = new SomeShape(...);
    // Draw shape
    g2d.draw(s); // outline
    g2d.fill(s); // solid
}
```

Java 2D Drawing Process: Step 1

- **Cast Graphics object to Graphics2D**

```java
public void paintComponent(Graphics g) {
    super.paintComponent(g); // Typical Swing
    Graphics2D g2d = (Graphics2D)g;
    g2d.doSomeStuff(...);
    ...
}
```

**Note**

- All methods that return Graphics in Java return Graphics2D in Java 2 and later
  - paint, paintComponent
  - getGraphics
Java 2D Drawing Process: Step 2

- **Set pen parameters**
  - `g2d.setPaint(fillColorOrPattern);`
  - `g2d.setStroke(penThicknessOrPattern);`
  - `g2d.setComposite(someAlphaComposite);`
  - `g2d.setFont(someFont);`
  - `g2d.translate(...);`
  - `g2d.rotate(...);`
  - `g2d.scale(...);`
  - `g2d.shear(...);`
  - `g2d.setTransform(someAffineTransform);`

Java 2D Drawing Process: Step 3

- **Create a Shape object.**
  - `Rectangle2D.Double rect = ...;`
  - `Ellipse2D.Double ellipse = ...;`
  - `Polygon poly = ...;`
  - `GeneralPath path = ...;`
  - `// Satisfies Shape interface`
  - `SomeShapeYouDefined shape = ...;`

- **Note**
  - Most shapes are in the `java.awt.geom` package
  - There is a corresponding Shape class for most of the `drawXxx` methods of Graphics (see next slide)
Built-in Shape Classes

- Arc2D.Double, Arc2D.Float
- Area (a shape built by union, intersection, subtraction and xor of other shapes)
- CubicCurve2D.Double, CubicCurve2D.Float
- Ellipse2D.Double, Ellipse2D.Float
- GeneralPath (a series of connected shapes), Polygon
- Line2D.Double, Line2D.Float
- QuadCurve2D.Double, QuadCurve2D.Float (a spline curve)
- Rectangle2D.Double, Rectangle2D.Float, Rectangle
- RoundRectangle2D.Double, RoundRectangle2D.Float
  - New shapes are in java.awt.geom. Java 1.1 holdovers (Rectangle, Polygon) are in java.awt. Several classes have similar versions that store coordinates as either double precision numbers (Xxx.Double) or single precision numbers (Xxx.Float). The idea is that single precision coordinates might be slightly faster to manipulate on some platforms.

Java 2D Drawing Process: Step 4

- Draw an outlined or filled version of the Shape
  - g2d.draw(someShape);
  - g2d.fill(someShape);

- The legacy methods are still supported
  - drawString still commonly used
  - drawLine, drawRect, fillRect still somewhat used
import javax.swing.*;  // For JPanel, etc.
import java.awt.*;      // For Graphics, etc.
import java.awt.geom.*; // For Ellipse2D, etc.

public class ShapeExample extends JPanel {
    private Ellipse2D.Double circle =
        new Ellipse2D.Double(10, 10, 350, 350);
    private Rectangle2D.Double square =
        new Rectangle2D.Double(10, 10, 350, 350);

    public void paintComponent(Graphics g) {
        clear(g);    // ie super.paintComponent(g);
        Graphics2D g2d = (Graphics2D)g;
        g2d.fill(circle);
        g2d.draw(square);
    }
    // Code to put JPanel in JFrame omitted.
}
Paint Styles in Java 2D: Overview

• **Idea**
  – Use setPaint and getPaint to change and retrieve the Paint settings.
  – Note that setPaint and getPaint supersede the setColor and getColor methods that were used in Graphics (and inherited in Graphics2D).
  – When you fill a Shape, the current Paint attribute of the Graphics2D object is used.

• **Possible arguments to setPaint are:**
  – A Color (solid color--Color implements Paint interface)
  – A GradientPaint (gradually-changing color combination)
  – A TexturePaint (tiled image)
  – A new version of Paint that you write yourself.
Paint Classes: Details

- **Color**
  - Has the same constants (Color.RED, Color.YELLOW, etc.) as the AWT version, plus some extra constructors.

- **GradientPaint**
  - Constructors take two points, two colors, and optionally a boolean flag that indicates that the color pattern should cycle. Colors fade from one color to the other.

- **TexturePaint**
  - Constructor takes a BufferedImage and a Rectangle2D, maps the image to the rectangle, then tiles the rectangle.
    - Creating a BufferedImage from a GIF or JPEG file is tedious. First load an Image normally, get its size, create a BufferedImage that size with BufferedImage.TYPE_INT_ARGB as the image type, and get the BufferedImage's Graphics object via createGraphics. Then, draw the Image into the BufferedImage using drawImage.

Simple Colors

- **In paint**
  ```java
  public void paint(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    g2d.setPaint(Color.RED);
    ...
  }
  ```

- **In paintComponent**
  ```java
  public void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    g2d.setPaint(Color.BLUE);
    ...
  }
  ```

- **From event handler**
  ```java
  public void mousePressed(MouseEvent e) {
    Graphics g = getGraphics();
    Graphics2D g2d = (Graphics2D)g;
    g2d.setPaint(Color.YELLOW);
    ...
  }
  ```
public class GradientPaintExample extends ShapeExample {
    private GradientPaint gradient =
        new GradientPaint(0, 0, Color.RED, 175, 175,
                           Color.YELLOW, true);
    // true means repeat pattern

    public void paintComponent(Graphics g) {
        clear(g);
        Graphics2D g2d = (Graphics2D)g;
        drawGradientCircle(g2d);
    }

    protected void drawGradientCircle(Graphics2D g2d) {
        g2d.setPaint(gradient);
        g2d.fill(getCircle());
        g2d.setPaint(Color.BLACK);
        g2d.draw(getCircle());
    }
}...
Tiled Images as Fill Patterns (TexturePaint): Overview

• **Step 1**
  – Create a TexturePaint object. TexturePaint constructor takes:
    • A BufferedImage (see following pages)
      – Specifies what to draw
    • A Rectangle2D
      – Specifies where tiling starts

• **Step 2**
  – Use the setPaint method of Graphics2D to specify that this TexturePaint object be used.
    • Applies to strings and outlines (i.e., draw operations), not just solid shapes (i.e., fill operations).

Creating a BufferedImage for Custom Drawing

• **Call the BufferedImage constructor with**
  – A width,
  – A height, and
  – A value of BufferedImage.TYPE_INT_RGB,

• **Call createGraphics on the result to get a Graphics2D that refers to image**
  – Use that Graphics2D object to draw onto the BufferedImage
Custom BufferedImage: Example Code

```java
int width = 32;
int height = 32;
BufferedImage bufferedImage =
    new BufferedImage(width, height,
        BufferedImage.TYPE_INT_RGB);
Graphics2D g2dImg = bufferedImage.createGraphics();
g2dImg.draw(...); // Draws onto image
g2dImg.fill(...); // Draws onto image
TexturePaint texture =
    new TexturePaint(bufferedImage,
        new Rectangle(0, 0, width, height));
g2d.setPaint(texture);
g2d.draw(...); // Draws onto window
g2d.fill(...); // Draws onto window
```

Creating a BufferedImage from an Image File

- **Quick summary**
  - Load an Image from an image file via `getImage`
  - Use `MediaTracker` to be sure it is done loading
  - Create an empty `BufferedImage` using the Image width and height
  - Get the `Graphics2D` via `createGraphics`
  - Draw the Image onto the `BufferedImage`

- **Utility class**
  - This process has been wrapped up in the `getBufferedImage` method of the `ImageUtilities` class
    - Like all examples, code available from URL on title page
**BufferedImage from Image File: Example Code**

```java
public class ImageUtilities {
    public static BufferedImage getBufferedImage
        (String imageFile, Component c) {
        Image image = c.getToolkit().getImage(imageFile);
        waitForImage(image, c); // Just uses MediaTracker
        BufferedImage bufferedImage =
            new BufferedImage(image.getWidth(c),
                               image.getHeight(c),
                               BufferedImage.TYPE_INT_RGB);
        Graphics2D g2dImg = bufferedImage.createGraphics();
        g2dImg.drawImage(image, 0, 0, c);
        return(bufferedImage);
    }
```

**Tiled Images as Fill Patterns: Example Code**

```java
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;

public class TiledImages extends JPanel {
    private String dir = System.getProperty("user.dir");
    private String imageFile1 = dir + "/images/marty.jpg";
    private TexturePaint imagePaint1;
    private Rectangle imageRect;
    private String imageFile2 = dir + "/images/bluedrop.gif";
    private TexturePaint imagePaint2;
    private int[] xPoints = { 30, 700, 400 };
    private int[] yPoints = { 30, 30, 600 };
    private Polygon imageTriangle =
        new Polygon(xPoints, yPoints, 3);
```
Tiled Images as Fill Patterns: Example Code (Continued)

```java
public TiledImages() {
    BufferedImage image =
        ImageUtilities.getBufferedImage(imageFile1, this);
    imageRect =
        new Rectangle(235, 70,
                     image.getWidth(), image.getHeight());
    imagePaint1 =
        new TexturePaint(image, imageRect);
    image =
        ImageUtilities.getBufferedImage(imageFile2, this);
    imagePaint2 =
        new TexturePaint(image,
                         new Rectangle(0, 0, 32, 32));
}
```

Tiled Images as Fill Patterns: Example Code (Continued)

```java
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;
    g2d.setPaint(imagePaint2);
    g2d.fill(imageTriangle);
    g2d.setPaint(Color.BLUE);
    g2d.setStroke(new BasicStroke(5));
    g2d.draw(imageTriangle);
    g2d.setPaint(imagePaint1);
    g2d.fill(imageRect);
    g2d.setPaint(Color.BLACK);
    g2d.draw(imageRect);
}
```
Transparent Drawing: Overview

• Idea
  – Assign transparency (alpha) values to drawing operations so that the underlying graphics partially shows through when you draw shapes or images.

• Normal steps
  – Create an AlphaComposite object
    • Call AlphaComposite.getInstance with a mixing rule designator and a transparency (or “alpha”) value.
    • There are 12 built-in mixing rules (see the AlphaComposite API for details), but you only care about AlphaComposite.SRC_OVER.
    • Alpha values range from 0.0F (completely transparent) to 1.0F (completely opaque).
  – Pass the AlphaComposite object to the setComposite method of the Graphics2D

Transparent Drawing: Example Code

```java
public class TransparencyExample extends JPanel {
    ...
    private AlphaComposite makeComposite(float alpha) {
        int type = AlphaComposite.SRC_OVER;
        return(AlphaComposite.getInstance(type, alpha));
    }

    private void drawSquares(Graphics2D g2d, float alpha) {
        Composite originalComposite = g2d.getComposite();
        g2d.setPaint(Color.BLUE);
        g2d.fill(blueSquare);
        g2d.setComposite(makeComposite(alpha));
        g2d.setPaint(Color.RED);
        g2d.fill(redSquare);
        g2d.setComposite(originalComposite);
    }
    ...
```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D)g;
    for(int i=0; i<11; i++) {
        drawSquares(g2d, i*0.1F); // 2nd arg is transparency
        g2d.translate(deltaX, 0);
    }
}
Using Logical (Java-Standard) Fonts

- **Logical font names**: use same names as in Java 1.1.
  - Serif (aka TimesRoman)
  - SansSerif (aka Helvetica -- results in Arial on Windows)
  - Monospaced (aka Courier)
  - Dialog
  - DialogInput.

Using Local (System-Specific) Fonts

- **Local fonts**: look up fonts first
  - Use the `getAvailableFontFamilyNames` or `getAllFonts` methods of `GraphicsEnvironment`. E.g.:
    ```java
    GraphicsEnvironment env =
        GraphicsEnvironment.getLocalGraphicsEnvironment();
    Then
    env.getAvailableFontFamilyNames();
    or
    env.getAllFonts();
    // Much slower than just getting names!
    
- **Safest option**: Supply list of preferred font names in order, loop down looking for first match. Supply standard font name as backup.
Example 1: Printing Out All Local Font Names

```java
import java.awt.*;

public class ListFonts {
    public static void main(String[] args) {
        GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
        String[] fontNames = env.getAvailableFontFamilyNames();
        System.out.println("Available Fonts:");
        for(int i=0; i<fontName.length; i++)
            System.out.println("  " + fontNames[i]);
    }
}
```

Example 2: Drawing with Local Fonts

```java
public class FontExample extends GradientPaintExample {
    public FontExample() {
        GraphicsEnvironment env = GraphicsEnvironment.getLocalGraphicsEnvironment();
        env.getAvailableFontFamilyNames();
        setFont(new Font("Goudy Handtooled BT", Font.PLAIN, 100));
    }

    protected void drawBigString(Graphics2D g2d) {
        g2d.setPaint(Color.BLACK);
        g2d.drawString("Java 2D", 25, 215);
    }

    public void paintComponent(Graphics g) {
        clear(g);
        Graphics2D g2d = (Graphics2D)g;
        drawGradientCircle(g2d);
        drawBigString(g2d);
    }
}
```
Drawing with Local Fonts: Example Output

Java 2D

Stroke Styles

Customized Java EE Training: http://courses.corejava.com/
Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.
Stroke Styles: Overview

• AWT
  – drawXX methods of Graphics resulted in solid, 1-pixel wide lines.
  – Predefined line join/cap styles for drawRect, drawPolygon, etc.

• Java2D
  – Pen thickness
  – Dashing pattern
  – Line join/cap styles

• Setting styles
  – Create a BasicStroke object
  – Use the setStroke method to tell the Graphics2D object to use it

Stroke Attributes

• Normal use: Use setStroke to assign a BasicStroke. BasicStroke constructors:
  – BasicStroke()
    • Creates a BasicStroke with a pen width of 1.0, the default cap style of CAP_SQUARE, and the default join style of JOIN_MITER.
  – BasicStroke(float penWidth)
    • Uses the specified pen width and the default cap/join styles.
  – BasicStroke(float penWidth, int capStyle, int joinStyle)
    • Uses the specified pen width, cap style, and join style.
  – BasicStroke(float penWidth, int capStyle, int joinStyle, float miterLimit)
    • Limits how far up the miter join can go (default is 10.0). Stay away from this.
  – BasicStroke(float penWidth, int capStyle, int joinStyle, float miterLimit, float[] dashPattern, float dashOffset)
    • Lets you make dashed lines by specifying an array of opaque (entries at even array indices) and transparent (odd indices) segments. The offset, which is often 0.0, specifies where to start in
import java.awt.*;

public class StrokeThicknessExample extends FontExample {
    public void paintComponent(Graphics g) {
        clear(g);
        Graphics2D g2d = (Graphics2D)g;
        drawGradientCircle(g2d);
        drawBigString(g2d);
        drawThickCircleOutline(g2d);
    }

    protected void drawThickCircleOutline(Graphics2D g2d) {
        g2d.setPaint(Color.BLUE);
        g2d.setStroke(new BasicStroke(8)); // 8-pixel wide pen
        g2d.draw(getCircle());
    }

    ...
}
public class DashedStrokeExample extends FontExample {
    public void paintComponent(Graphics g) {
        clear(g);
        Graphics2D g2d = (Graphics2D)g;
        drawGradientCircle(g2d);
        drawBigString(g2d);
        drawDashedCircleOutline(g2d);
    }

    protected void drawDashedCircleOutline(Graphics2D g2d) {
        g2d.setPaint(Color.BLUE);
        // 30 pixel line, 10 pxl gap, 10 pxl line, 10 pxl gap
        float[] dashPattern = { 30, 10, 10, 10 };
        g2d.setStroke(new BasicStroke(8, BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_MITER, 10,
                dashPattern, 0));

        g2d.draw(getCircle());
    }
    ...
}
Join Styles

• **JOIN_MITER**
  - Extend outside edges of lines until they meet
    - This is the default

• **JOIN_BEVEL**
  - Connect outside corners of outlines with straight line

• **JOIN_ROUND**
  - Round off corner with a circle that has diameter equal to the pen width

Cap Styles

• **CAP_SQUARE**
  - Make a square cap that extends past the end point by half the pen width
    - This is the default

• **CAP_BUTT**
  - Cut off segment exactly at end point
    - Use this one for dashed lines.

• **CAP_ROUND**
  - Make a circle centered on the end point. Use a diameter equal to the pen width.
public class LineStyles extends JPanel {
    private int[] caps =
    { BasicStroke.CAP_SQUARE, BasicStroke.CAP_BUTT, BasicStroke.CAP_ROUND };
    private int[] joins =
    { BasicStroke.JOIN_MITER, BasicStroke.JOIN_BEVEL, BasicStroke.JOIN_ROUND };

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D)g;
        g2d.setColor(Color.BLUE);
        for(int i=0; i>caps.length; i++) {
            BasicStroke stroke =
                new BasicStroke(thickness, caps[i], joins[i]);
            g2d.setStroke(stroke);
            g2d.draw(path);
        }
    }
}
Coordinate Transformations

- **Idea:**
  - Instead of computing new coordinates, move the coordinate system itself.

- **Available Transformations**
  - Translate (move).
  - Rotate (spin).
  - Scale (stretch evenly)
  - Shear (stretch more as points get further from origin)
  - Custom. New point (x2, y2) derived from original point (x1, y1) as follows:

\[
\begin{bmatrix}
  x2 \\
  y2 \\
  1
\end{bmatrix}
= \begin{bmatrix}
  m00 & m01 & m02 \\
  m10 & m11 & m12 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x1 \\
  y1 \\
  1
\end{bmatrix}
= \begin{bmatrix}
  m00x1 + m01y1 + m02 \\
  m10x1 + m11y1 + m12 \\
  1
\end{bmatrix}
\]
public class RotationExample
   extends StrokeThicknessExample {
   private Color[] colors = { Color.WHITE, Color.BLACK };

   public void paintComponent(Graphics g) {
      clear(g);
      Graphics2D g2d = (Graphics2D)g;
      drawGradientCircle(g2d);
      drawThickCircleOutline(g2d);
      // Move the origin to the center of the circle.
      g2d.translate(185.0, 185.0);
      for (int i=0; i<16; i++) {
         // Rotate the coordinate system around current
         // origin, which is at the center of the circle.
         g2d.rotate(Math.PI/8.0);
         g2d.setPaint(colors[i%2]);
         g2d.drawString("Java", 0, 0);
      }
   }
}
Shear Transformations

- **Meaning of Shear**
  - **X shear**
    If you specify a non-zero x shear, then x values will be more and more shifted to the right the farther they go in the y direction. For example, an x shear of 0.1 means that the x value will be shifted 10% of the distance the point is away from the x axis.
  - **Y shear**
    Points are shifted down in proportion to the distance they are to the right (x direction) from the y axis.

- **Note**
  - Called “skew” instead of “shear” in some graphics libraries (e.g., with Android drawing)

---

Shear: Example Code

```java
public class ShearExample extends JPanel {
    private static int gap=10, width=100;
    private Rectangle rect =
        new Rectangle(gap, gap, 100, 100);

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D)g;
        for (int i=0; i<5; i++) {
            g2d.setPaint(Color.RED);
            g2d.fill(rect);
            // Each new square gets 0.2 more x shear
            g2d.shear(0.2, 0.0);
            g2d.translate(2*gap + width, 0);
        }
    }
}
```
Shear: Example Output

Shear: x shear ranges from 0.0 for the leftmost 'square' to 0.8 for the rightmost one.

Drawing Accuracy
Rendering Hints

- **Default:**
  - Faster drawing, possibly less accuracy

- **Rendering Hints:**
  - Let you request more accurate (but generally slower) drawing. Eg:
    ```java
    RenderingHints renderHints =
    new RenderingHints(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON);
    renderHints.put(RenderingHints.KEY_RENDERING,
                     RenderingHints.VALUE_RENDER_QUALITY);
    ```

    ```java
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D)g;
        g2d.addRenderingHints(renderHints);
    }
    ```
Summary

• General
  – If you have Graphics, cast it to Graphics2D
  – Create Shape objects, then call Graphics2D’s draw and
    fill methods with shapes as args.

• Paint styles
  – Use setPaint to specify a solid color (Color), a gradient
    fill (GradientPaint), or tiled image (TexturePaint).
    TexturePaint requires a BufferedImage, which you can
    create from an image file by creating empty
    BufferedImage then drawing image into it.

• Transparent drawing
  – Use AlphaComposite for transparency. Create one via
    AlphaComposite.getInstance with a type of
    AlphaComposite.SRC_OVER.

Summary (Continued)

• Local fonts
  – Look up locally installed fonts with
    getAvailableFontFamilyNames. Then supply name to
    Font constructor and specify font via setFont.

• Stroke styles
  – BasicStroke lets you set pen thickness, dashing pattern,
    and line cap/join styles. Then call setStroke.

• Coordinate transformations
  – Let you move the coordinate system rather than changing
    what you draw. Simple transforms: call translate, rotate,
    scale, and shear. More complex transforms: supply matrix
    to AffineTransform constructor, then call setTransform.

• Rendering Hints
  – Improve drawing quality or enable antialiasing
Questions?

More info:
http://courses.coreservlets.com/Course-Materials/java.html – General Java programming tutorial
http://www.coreservlets.com/java-8-tutorial/ – Java 8 tutorial
http://courses.coreservlets.com/java-training.html – Customized Java training courses, at public venues or onsite at your organization
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Services, JSP 2.0, and other Java EE training

Customized Java EE Training: http://courses.coreservlets.com/
Java 7, Java 8, JSF 2.2, PrimeFaces, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.